

REMARKS

Claims 1-28 are currently pending, of which claims 1, 13, 25, 26, and 28 are independent. All claims have been rejected under 35 U.S.C. § 103(a). These rejections are respectfully traversed. Reconsideration is requested.

Rejections under 35 U.S.C. § 103(a)

Claims 1-7, 10-16, and 18-28 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over McLennan, Michael J., "Object-oriented Programming with [incr Tcl] Building Mega-Widget with [incr Tk]," (art of record (AV), hereinafter McLennan) in view of Tk Library Procedures, "Tk-configure Widget Manual Page," (art of record (AR2), hereinafter TkLib). Claims 8-9, and 17 stand rejected under U.S.C. § 103(a) as being unpatentable over McLennan in view of TkLib, further in view of Hostetter et al., "Curl: A Gentle Slope Language for the Web," (art of record (AS), hereinafter Hostetter et al.).

In order to establish a prima facie case of obviousness under 35 U.S.C. § 103(a), three criteria must be met. There must be: (i) some suggestion or motivation to modify the reference or to combine reference teachings; (ii) a reasonable expectation of success; and (iii) a teaching or suggestion of all the limitations of the claims. See M.P.E.P. § 2143. For the reasons discussed below, it is respectfully submitted that the Examiner has not established a prima facie case of obviousness for any of the present claims, and that therefore, the present claims are allowable.

Briefly describing the present invention, preferred embodiments of the invention relate to a technique that allows type processing of option values during compilation without having to store full option values in memory. A class is defined that supports an option data structure. An instance of the class has references to option values without preallocating memory space for full option values. In fact, claim 1 recites in pertinent part "... defining a class which supports an option data structure . . . without preallocation of memory space for the full option values." Exemplary option data structures include a linked list or a hash table. Storage space is only allocated when the default value for an option value is modified for the object in contrast to prior art methods that allocate memory for an attribute even when the object inherits a value for the attribute from an object up the hierarchy. Further reciting from claim 1, "during compilation, [the compiler uses] the type descriptions in the option data structure to process an operation on

the option values.” In this manner, the invention enables compile-time processing of options with type information, but without preallocated storage.

By way of contrast, the previously-amended background section of the Applicants’ specification discusses prior art techniques for storing property values that relate to the [incr Tk] language. In [incr Tk], there is preallocation of memory space for option values. [incr Tcl] is a Tcl extension that adds object-oriented programming constructs to Tcl. [incr Tk] builds on top of a graphics toolkit called Tk, so when an application is built using [incr Tk], [incr Tk] “mega-widgets” may be used which may in turn contain other [incr Tk] mega-widgets, and may also contain basic Tk widgets. Tk defines the basic widgets such as text-entry fields and scrollbars, while the [incr Tk] mega-widgets are larger-scale entities such as file choosers or color choosers.

Turning briefly to the cited references, in turn, McLennan first describes the Tcl language as being interpretive in nature. In particular, McLennan indicates that the speed of development of applications in Tcl/Tk is partly “due to the interpretive nature of the Tcl language; changes made to a Tcl/Tk application can be seen immediately, without waiting for the usual compile/link/run cycle.” (McLennan, page 1, lines 6-8 of the Abstract). Thus, the Tcl language that is used to write applications using the Tk and [incr Tk] toolkits is interpreted, not compiled. McLennan also discusses the construction of a mega-widget. (See pages 76-85.) As shown in Fig. 2-8, each mega-widget includes a set of configuration options. A configuration option can have the same value as the option on a master list or can be set just for the mega-widget. Each widget class has a default set of options that have the same value as the option on a master list. The default set of options is called the usual option-handling code. For example, the foreground, background, font and cursor options in a label component have the same values as the options on the master list (i.e., using the normal Tk widgets as component parts). (See page 83, lines 5-35.)

Therefore, it is pertinent to look at the storage requirements both of Tk widgets and [incr Tk] mega-widgets. These are discussed in the manual pages for each release of Tk. The Tklib reference provides manual pages for the basic Tk widget (see the release 8.0.x manual pages, released between August 1997 and March 1999, available at: <http://www.tcl.tk/man/tcl8.0/TkLib/ConfigWidg.htm>). Referring to the Tklib reference, Tk_ConfigureWidget processes a table specifying configuration options that are supported. Each entry in the table is a Tk_ConfigSpec structure, which includes a type field (type of configuration

option, for example, color) and an offset field indicating where in the record to store information about this option. (see Tklib, last two paragraphs of page 3 of 9).

In particular, in the middle of page 7 of 9, under the heading “TK_OFFSET,” there is the following paragraph:

The Tk_Offset macro is provided as a safe way of generating the offset values for entries in Tk_ConfigSpec structures. It takes two arguments: the name of a type of record, and the name of a field in that record. It returns the byte offset of the named field in records of the given type.

From this paragraph (and the surrounding context), it can be inferred that every option value that can be set on a Tk widget has a unique, pre-allocated location in the widget's “record” where that option value will be stored. In addition, to the above-paragraph describing the Tk_Offset macro, the man page also describes a Tk_ConfigureWidget macro that includes “a collection of command-line arguments (argc and argv) to fill in fields of a record (widgRec).” It can be inferred that in order to fill in the fields, the fields must be pre-allocated. Furthermore, the Tk_ConfigSpec structure that specifies a configuration option includes an offset field which “indicates where in widgRec to store information about this option.” Thus, space is pre-allocated for every option value, before any options are actually set. If space were not pre-allocated, but rather allocated only for options that are actually set, it would not be possible to specify a fixed offset in the record for storing a value simply based on knowing the type of record and the name of a field. It would also be necessary to take into account the other options that had already been set on the specific record in question (i.e., different selected options for a given record would result in different storage offsets). Details as to any other options that might be associated with the record would not be provided by the type of record and the name of a field alone.

Further detail as to the storage requirements of [incr Tk] mega-widgets can be found at the man page for Archetype (Archetype is the base class for all [incr Tk] mega-widgets). For example, the man page for Archetype in version 3.1 of [incr Tk] (released May 29, 1999) is available on the Internet at [http:// www.tcl.tk/man/itcl3.1/Archetype.n.html](http://www.tcl.tk/man/itcl3.1/Archetype.n.html) (Art of record (AS2)). The “itk_component add” method is used to add a component widget to a mega-widget. In calling this method, the “keep” command (documented under “itk_component add”) is used to specify option names of the component widget that should be connected to option names of

the mega-widget. Each option name to be “kept” is specified individually, and later on if one of these options is set on the mega-widget, the [incr Tk] mechanisms will notice that and will also set the same option value on the component widget. Since each mega-widget instance will have its own collection of component widget instances, it must be the case that each mega-widget instance builds up its own table of “kept” options, whose size is (at a minimum) proportional to the number of options that could be set on the mega-widget, regardless of how many of those options are actually set on the mega-widget.

The “itk_option define” command provides a mechanism for defining option names on mega-widgets that might not correspond to option names on any of the component widgets. The arguments to this command include an “init value” for which the following explanation is provided: “The init value string is used as a last resort to initialize the option if no other value can be used from an existing option, or queried from the X11 resource database.” The discussion of “initializing” the option value from the “init” argument provides strong evidence that there is already a preallocated memory location to hold the option value: given the existence of this location, it must be filled with some value, so the init value is used as a last resort for this purpose. If a preallocated location did not already exist, there would be no discussion of the need for “initializing.” In contrast, there may be a discussion regarding use of default value if the location was not allocated.

With respect to base claim 1, the option data structures are explicitly claimed as being applied “during compilation, using the type description in the option data structure to process an operation on the option value.” Applicants respectfully submit that the Office Action mischaracterizes McLennan by stating that it discloses “during compilation, using the type description in the option data structure . . .” (the Office Action specifically refers to pages 76 and 79 of McLennan as providing an example of doing this). Inconsistent with the above statement, the Office Action later states that “McLennan does not explicitly disclose his teaching type description of the option values and during compilation, using the type description in the option data structure to process an operation on the option value.” The latter statement appears to be more accurate, at least with respect to using the type description during compilation. As described above, TCL language is interpreted rather than compiled. Thus, McLennan could not accomplish anything during compilation, because compilation does not occur. The Tklib

reference fails to cure this deficiency, as neither Tk nor [incr Tk] perform compilation. The Tk toolkit as described in the Tklib reference does not perform any compilation operations based on option data structures. Instead, the Tk and [incr Tk] option data structures are used only by run-time libraries, chiefly the Tk_ConfigureWidget subroutine documented in the Tklib reference.

Using type information in the option data structure during compile time to process operations on option values offers several advantages. First, a compiler can use the type information to check the legality of option operations in a program. (See for example, page 2, lines 5-9). For example, if “width,” “color,” and “enabled” are options, a compiler can determine that operations such as:

```
set x.width = 10cm;
set x.color = “red”; and
set x.enabled = true
```

are legal, while determining that operations such as:

```
set x.width = “purple”;
set x.color = 2.54cm; and
set x.enabled = “Minnesota”
```

are illegal. Such early reporting of illegal operations at the time when a program is compiled provides early warning about programming bugs. (See for example, page 5, line 18 through page 6, line 2). In contrast, using the Tk and [incr Tk] approach, no warning would be provided until the program actually runs and attempts to execute an offending statement. Second, a compiler can use the type information to generate more efficient code. (See for example, page 11, lines 2-4). For example, knowing that the “enabled” option is a Boolean-valued (true or false) option, a compiler can generate optimized code that uses only one bit for the Boolean value, instead of using a more general and expensive data representation. These advantages of compile-time processing of option type information, and the technology required to implement such processing, are both significant steps beyond anything taught in the Tk or [incr Tk] prior art.

Thus, McLennan in combination with the Tklib reference fail to establish a prima facie obviousness as neither reference considered alone or in combination teaches nor suggests the claimed technique “during compilation, [using] the type descriptions in the option data structure to process an operation on the option values.” To the contrary, McLennan does not involve

compilation as it is interpretive in nature and the Tklib reference fails to cure the defect as the option data structures described therein are used by run-time libraries.

Additionally, with respect to base claim 1, the Office Action states that McLennan discloses a method of “defining a class which supports an option data structure . . . without preallocation of memory space for the full option value.” Applicants respectfully submit, however, that the Examiner’s characterization of McLennan, at least with respect to the preallocation of memory, is also inaccurate. As described above, the mega-widgets of McLennan use the normal Tk widgets as component parts. Because, McLennan’s mega-widgets use normal Tk widgets as components, they also require a preallocation of memory space for option values as suggested above. Thus, McLennan fails to teach or suggest, at least, “defining a class which supports an option data structure . . . without preallocation of memory space.”

The Tklib reference fails to cure the deficiencies of McLennan. As described above, the “Tk_Offset macro” of the Tklib reference is to generate byte offset values for entries in Tk_ConfigSpec structures. Because the offset macro returns the byte offset of a named field in records using only the name of a type of record and the name of a field in that record, the storage space for every option value must be pre-allocated before any options are actually set.

Further, it would not have been obvious to modify either McLennan or the Tklib reference to define a class which supports an option data structure without preallocating memory space for the full option value, at least because the Tk_Offset macro would be unable to provide a suitable byte offset. Without knowing the storage space for every option value, a byte offset for one option could be provided that interferes with another option for the same record. In fact, without being assured that an option can be selected on a non-interfering basis, there would be little likelihood of success.

Thus, McLennan in combination with the Tklib reference fail to establish a prima facie obviousness as neither reference considered alone or in combination teaches nor suggests the claimed technique for defining an object with an option data structure that supports references to option values without preallocation of memory for full option values. Rather, it appears that McLennan is consistent with traditional object-oriented techniques, as McLennan does not suggest modifying them in any way. As a result, a preallocated field is provided for each of the configuration options when the object is created, even if the option is never set on the object.

Thus, even when this option is not set, it still occupies its own dedicated memory space in the object.

As dependent claims 2-7 and 10-12 depend either directly or indirectly from base claim 1, they also contain all of the limitations of the base claim. Accordingly, claims 2-7 and 10-12 are non-obvious for the same reasons argued above in relation to base claim 1.

Base claims 13, 25, 26, and 28 and their respective dependent claims 14-16, 18-24 and 27 contain similar limitations to base claim 1 and are therefore non-obvious for the same reasons as argued above in relation to claim 1.

Claims 8-9 and 17 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over McLennan in view of TkLib in claims 1 and 14, in further view of Hostetter et al., "Curl: A Gentle Slope Language for the Web," (art of record (AS), hereinafter Hostetter et al.).

Hostetter et al. describes the design of Curl, a single, coherent linguistic basis for expression of Web content at levels ranging from simple formatted text to contemporary object-oriented programming. Hostetter et al. describes using consistent syntax and semantics for the expression of Web content at levels ranging from simple formatted text to contemporary object-oriented programming. (Curl, page 2 of 14, lines 22-23). Curl is further described as both a language and an authoring environment, using object representations similar to those of C++. Because Curl is similar to other standard object-oriented techniques, Hostetter et al. does not teach or suggest "defining a class which supports an option data structure . . . without preallocation of memory space." Accordingly, Hostetter et al. fails to cure the deficiencies of McLennan and the Tklib reference.

As claims 8-9 depend either directly or indirectly from claim 1, they also contain all of the limitations of base claim 1. Similarly, as claim 17 depends directly from base claim 13, it contains all of the limitations of that base claim. Because Curl fails to correct the deficiencies of McLennan and the Tklib reference, the Office Action fails to establish prima facie obviousness for claims 8-9 and 17 for at least the same reasons argued above in relation to base claims 1 and 13.

For the reasons set forth above, Applicants submit that claims 1, 13, 25, 26, and 28 and their respective dependent claims are in condition for allowance. Reconsideration of the rejections under 35 U.S.C. § 103(a) is respectfully requested.

Information Disclosure Statement

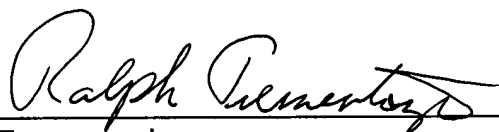
An Information Disclosure Statement (IDS) is being filed concurrently herewith. Entry of the IDS is respectfully requested.

CONCLUSION

In view of the above amendments and remarks, it is believed that all claims are in condition for allowance, and it is respectfully requested that the application be passed to issue. If the Examiner feels that a telephone conference would expedite prosecution of this case, the Examiner is invited to call the undersigned.

Respectfully submitted,

HAMILTON, BROOK, SMITH & REYNOLDS, P.C.

By 
Ralph Trementozzi
Registration No. 55,686
Telephone: (978) 341-0036
Facsimile: (978) 341-0136

Concord, MA 01742-9133

Dated:

9-16-2004